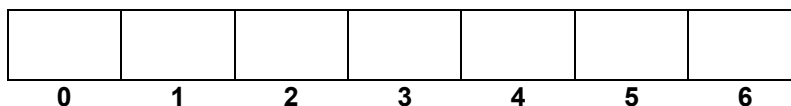


ARREGLOS Y MATRICES

1. Arreglos

Un arreglo es una estructura de datos, o más técnicamente, un espacio de memoria que permite almacenar una colección de elementos, todos del mismo tipo. Conviene imaginar un arreglo como una secuencia contigua de celdas (espacios de memoria), o **casillas**, en cada una de las cuales se puede guardar un elemento de la colección. Además, es usual dibujarlo como lo ilustra la figura siguiente:



Esta figura representa un arreglo de siete casillas cada una de las cuales se puede utilizar para guardar un dato. La **dimensión** o tamaño de un arreglo es el número de casillas que lo conforman. Debe ser claro, entonces, que la figura anterior corresponde a un arreglo de dimensión 7.

Cada una de las casillas de un arreglo tiene asociado un número que la identifica de manera única. A este número se le llama **índice** o **dirección**. En la figura anterior, debajo de cada casilla, aparece su índice. En lenguajes como C, C++ y java, la primera casilla del arreglo tiene índice 0, la segunda tiene índice 1, la tercera índice 2, y así sucesivamente. Es muy importante tener presente que si el arreglo es de dimensión **N**, la última casilla tiene índice **N-1**.

Los lenguajes de programación, permiten que el programador declare arreglos de cualquier tipo y prácticamente de cualquier tamaño. En el pseudolenguaje, un arreglo se declara usando el siguiente formato o plantilla:

<NOMBRE> : arreglo [<N>] de <TIPO>

En este formato aparecen en mayúsculas y entre los caracteres < y > los componentes que el programador debe determinar. Así por ejemplo, si se quiere declarar un arreglo con nombre **letras**, de dimensión **15** y que pueda almacenar datos de tipo **caracter**, se debe escribir la siguiente línea.

letras : arreglo [15] de caracter

Volviendo al formato anterior, el programador debe bautizar el arreglo (ponerle un nombre significativo), debe decir cuál es su dimensión, y también debe decir de qué tipo son los elementos que almacenará ese arreglo.

Enseguida se dan algunos ejemplos de declaraciones de arreglos.

- Si se necesita guardar las ventas diarias de una tienda durante la última semana, se puede declarar el siguiente arreglo:

ventas : arreglo [7] de real

- Si se quiere guardar las notas que ha sacado un estudiante en los cinco talleres y en los cinco laboratorios del curso de Programación de Computadores se pueden declarar los siguientes arreglos:

talleres : arreglo [5] de real

laboratorios : arreglo [5] de real

- Si se quiere guardar el valor de las últimas 12 facturas telefónicas de una casa, se puede declarar el siguiente arreglo:

facturasTel : arreglo [12] de real

Los índices se crearon para permitir que el programador se pueda referir, de forma específica, a una cualquiera de las casillas del arreglo, tanto para guardar un dato en esa casilla, como para obtener el dato guardado. Para referirse a una casilla particular de un arreglo se debe seguir el siguiente formato:

<NOMBRE>[<INDICE>]

es decir, se debe escribir el nombre del arreglo seguido por el índice de la casilla entre paréntesis cuadrados.

Para los siguientes ejemplos, suponga que se declara el arreglo **cifras**, de la siguiente manera:

cifras : arreglo [10] de entero

- La siguiente instrucción asigna o guarda el número 100 en la primera casilla de este arreglo:

cifras[0]:= 100

- La siguiente instrucción iterativa guarda 550 en cada una de las últimas 5 casillas de este arreglo:

i:=5

MIENTRAS (i<10) HACER

cifras[i]:= 550

i:=i+1

FIN-MIENTRAS

La siguiente figura muestra el arreglo **cifras** después de ejecutadas las instrucciones de los dos ejemplos anteriores. Las casillas vacías no tienen valores definidos.

100					550	550	550	550	550
0	1	2	3	4	5	6	7	8	9

1.1 Ejemplo completo

Un **histograma** para una colección de datos es una secuencia de parejas de la forma **(d,f)**, donde **d** es un dato y **f** es su frecuencia en la colección. Por ejemplo, suponga que se le pide a 20 personas calificar con las letras **a b c d** y **e** el desempeño del gobierno actual, y que se obtienen las siguientes respuestas: **c b c a b c d e e a b b d c a c c b d a**. El histograma para esta colección de datos se muestra enseguida en dos formas: con números y con asteriscos:

a: 4	a: ****
b: 5	b: *****
c: 6	c: *****
d: 3	d: ***
e: 2	e: **

Considere el problema de construir un algoritmo que haga un histograma para una lista de hasta 100 valores, donde cada valor es un número entero comprendido en el intervalo 1 al 5.

Las entradas (datos conocidos) para el algoritmo son:

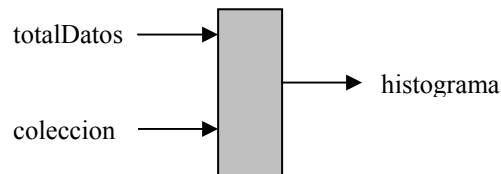
- El número de datos de la colección
- La colección misma

La salida esperada (dato desconocido) es:

- El histograma de la colección

En este problema, los arreglos son útiles para guardar los datos que conforman la colección y también para guardar el histograma. El número de datos de la colección se puede guardar en una variable entera.

La siguiente gráfica resume las entradas y las salidas del algoritmo que se pretende diseñar. Además bautiza todas las variables mencionadas:



Las condiciones iniciales y finales se pueden expresar mediante dos cláusulas: **REQUIERE** y **GARANTIZA**, de la siguiente manera:

REQUIERE:

El número de datos de la colección debe ser mayor que cero y menor o igual que cien. ($100 \geq \text{totalDatos} > 0$)

Cada uno de los elementos de la colección debe ser un número entre 1 y 5.

GARANTIZA:

Calcula el histograma asociado a la colección y lo presenta en la pantalla. La frecuencia de cada dato aparece graficado como una secuencia de asteriscos.

Una primera versión del algoritmo solución puede ser simplemente la siguiente:

Inicio

Paso 1. Leer el número de elementos que tiene la colección

Paso 2. Leer los elementos de la colección

Paso 3. Calcular el histograma

Paso 4. Presentar el histograma en la pantalla

Fin

Los pasos 1 y 2 son interacciones con el usuario que permiten capturar los datos de entrada. La versión inicial se puede refinar detallando estos pasos y además, definiendo las variables necesarias para hacerlos:

Procedimiento principal**Variables**

i, totalDatos: entero

colección: arreglo [100] de entero

Inicio

escribir("Por favor digite el número de datos de la colección : (inferior o igual a 100)")

leer(totalDatos)

i:=0

mientras (i<totalDatos) hacer

escribir("Por favor digite el dato :")

escribir(i+1)

leer(colección[i])

i:= i+1

fin-mientras

Paso 3. Calcular el histograma

Paso 4. Presentar el histograma en la pantalla

Fin

La parte nuclear de la solución es el Paso 3 (calcular el histograma). En este problema particular se sabe que todos los datos están entre 1 y 5, lo cual quiere decir que se necesita calcular cinco frecuencias: la del 1, la del 2, etc. Es natural entonces usar un arreglo de 5 casillas para guardar el histograma, de tal manera que en la casilla 0 estará la

frecuencia del 1, en la casilla 1 estará la frecuencia del 2, y así sucesivamente. A esta variable se le llamará “histograma” y su declaración es:

histograma: arreglo[5] de entero

Ahora, para calcular las frecuencias se puede hacer lo siguiente:

Paso 3.1 inicializar las cinco casillas del arreglo histograma en cero

Paso 3.2

revisar la colección, dato por dato, e ir incrementando la frecuencia asociada a cada uno de ellos. Así, si se encuentra un 5, se debe incrementar en 1 la frecuencia asociada al 5, es decir, se le debe sumar 1 a histograma[4].

Enseudolenguaje, el cálculo del histograma puede ser:

```
i:=0
mientras (i<5) hacer      // inicializa las frecuencias en 0. Paso 3.1
    histograma[i]:= 0
    i:= i+1
fin-mientras
i:=0
mientras (i<totalDatos) hacer // calcula las frecuencias. Paso 3.2
    d:= colección[i]-1
    histograma[d]:= histograma[d]+1
    i:= i+1
fin-mientras
```

Finalmente, una vez calculado el histograma, se debe presentar en la pantalla. Cada dato se debe presentar junto con su frecuencia, pero se requiere que la frecuencia aparezca como una cadena de asteriscos, en vez de como un número. Por ejemplo, si el dato 8 aparece 3 veces en la colección, en la pantalla debe aparecer 8: ***, como una línea del histograma. Esto quiere decir que para escribir la frecuencia de un dato, se requiere un ciclo que escriba tantos asteriscos como sea la frecuencia.

Concluyendo, el paso 4 (presentar el histograma en la pantalla) se puede refinar como se muestra enseguida:

```
i:=0
mientras (i<5) hacer      // este ciclo recorre el histograma
    escribir(i+1)
    escribir(": ")
    f:= histograma[i]      // f guarda la frecuencia de dato i+1
    j:= 0
    mientras(j<f) hacer    // este ciclo escribe f asteriscos
        escribir ("*")
        j:= j+1
    fin-mientras
    escribir(salto-de-linea)
```

```
        i:= i+1
fin-mientras
```

El algoritmo completo se presenta enseguida. Se han definido algunas constantes para permitir que el programa sea más fácilmente modificable.

Procedimiento principal

Constantes

```
N 5           / N es el tamaño máximo del histograma
MAXDATOS 100  / MAXDATOS es el tamaño máximo de la colección
```

Variables

```
i,j,totalDatos: entero
coleccion: arreglo [MAXDATOS] de entero
histograma: arreglo [N] de entero
```

Inicio

```
escribir("Por favor digite el número de datos de la colección : (inferior o igual a
100)")
leer(totalDatos)
i:=0
mientras (i<totalDatos) hacer
    escribir("Por favor digite el dato :")
    escribir(i+1)
    leer(colección[i])
    i:= i+1
fin-mientras
i:=0
mientras (i<N) hacer    // inicializa las frecuencias en 0. Paso 3.1
    histograma[i]:= 0
    i:= i+1
fin-mientras
i:=0
mientras (i<totalDatos) hacer // este ciclo calcula las frecuencias. Paso 3.2
    d:= colección[i]-1
    histograma[d]:= histograma[d]+1
    i:= i+1
fin-mientras
i:=0
mientras (i<N) hacer    // este ciclo recorre el histograma
    escribir(i+1)
    escribir(": ")
    f:= histograma[i]    // f guarda la frecuencia de dato i+1
    j:= 0
    mientras(j<f) hacer    // este ciclo escribe f asteriscos
        escribir ("*")
        j:= j+1
```

```

    fin-mientras
    escribir(salto-de-linea)
    i:= i+1
fin-mientras
Fin

```

1.2 Problemas

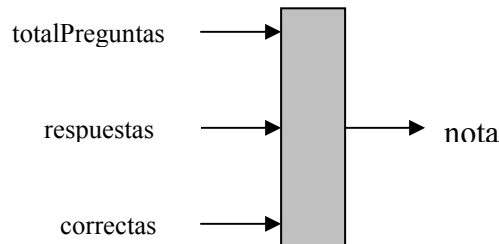
1. Suponga que se quiere construir un algoritmo que califique un examen de selección múltiple presentado por un estudiante de Programación de Computadores. En cada una de las preguntas del examen, el estudiante debió elegir una de cinco opciones, identificadas con las letras A,B,C,D y E. Las entradas (datos conocidos) para el algoritmo son:

- El número de preguntas que tenía el examen
- Cada una de las respuestas dadas por el estudiante
- Las respuestas correctas

La salida esperada (dato desconocido) es:

- la nota obtenida. Esta nota corresponde al número de aciertos que tuvo el estudiante.

En este problema, los arreglos son útiles para guardar las respuestas correctas y las opciones elegidas por el estudiante. El número de preguntas del examen se puede guardar en una variable entera, al igual que la nota. Enseguida se muestra la especificación de este problema:



REQUIERE:

El número de preguntas del examen debe ser mayor que cero. (**totalPreguntas>0**)

Cada una de las respuestas del estudiante debe ser una letra mayúscula que está entre **A** y **E**.

Cada una de las respuestas correctas debe ser una letra mayúscula que está entre **A** y **E**.

GARANTIZA:

La nota dada por el algoritmo corresponde al total de respuestas acertadas del estudiante

Escriba un algoritmo que cumpla con esta especificación.

2. Escriba un algoritmo que efectúe la normalización de una colección de números reales. Para llevar a cabo esta normalización, se debe en primer lugar encontrar el número mayor de la colección; luego se divide cada número por dicho valor máximo, de forma que los valores resultantes (normalizados) estén comprendidos en el intervalo del 0 al 1.
3. Escriba un algoritmo que sume en binario. Las entradas son dos números (binarios) y la salida es la suma de estos dos valores (también en binario). Por ejemplo, si el usuario digita las cadenas binarias **101** y **1101**, la respuesta dada debe ser **10010**.

1.3 Ejercicios

1. Escriba un algoritmo que lea dos arreglos de números enteros ORDENADOS ascendentemente y luego produzca la lista ordenada de la mezcla de los dos. Por ejemplo, si los dos arreglos tienen los números **1 3 6 9 17** y **2 4 10 17**, respectivamente, la lista de números en la pantalla debe ser **1 2 3 4 6 9 10 17 17**.
2. Escriba un algoritmo que lea un arreglo de números enteros, y un número **x**, y escriba en la pantalla todos los índices de las posiciones del arreglo donde está **x**. Por ejemplo, si el arreglo es el que aparece enseguida y **x** es **2**:

1	2	3	100	23	2	2	1
---	---	---	-----	----	---	---	---

El programa debe escribir: **1 5 6**.

3. Un arreglo de números se llama **partidario** si todo número que está en una casilla par (0,2,4,...) es mayor que cualquiera de los números que están en las casillas impares (1,3,5,...). Escriba un algoritmo que lea un arreglo de números enteros y luego, diga si es partidario o no. Por ejemplo, si el arreglo es el siguiente:

100	5	200	1	1000	0	600	50	300	4
-----	---	-----	---	------	---	-----	----	-----	---

El programa debe escribir: **es partidario**.

2. Cadenas de caracteres

Los elementos del tipo **caracter** (tipo **char** en lenguaje **C**) se pueden agrupar para formar secuencias que se denominan **cadenas de caracteres**, o simplemente **cadenas**. En este texto, y también en el texto de un programa en **C**, las cadenas se delimitan por dobles comillas. Por ejemplo, **"CH?*\$A7!"** y **"soy cadena"** son dos cadenas, la primera formada por 8 caracteres y la segunda por 10. En la memoria del computador una cadena se guarda en un arreglo de tipo **caracter**, de tal manera que cada símbolo de la cadena ocupa una casilla del arreglo. Sin embargo, se utiliza una casilla adicional del arreglo para guardar un carácter especial que se llama **terminador de cadena**. En **C** y en elseudolenguaje este carácter especial es **'\0'**. Como lo indica su nombre, la función de este carácter especial es indicar que la cadena termina. Las dos cadenas mencionadas arriba, se representan en la memoria del computador como lo indica la siguiente figura:

'C'	'H'	'?'	'*'	'\$'	'A'	'7'	'!'	'\0'		
0	1	2	3	4	5	6	7	8		

's'	'o'	'y'	' '	'c'	'a'	'd'	'e'	'n'	'a'	'\0'
0	1	2	3	4	5	6	7	8	9	10

La **longitud** de una cadena se define como el número de símbolos que la componen, sin contar el terminador de cadena. Es muy importante tener en cuenta que aunque el terminador de cadena no hace parte de la cadena, sí ocupa una casilla de memoria en el arreglo.

Dado que la representación de cadenas es mediante arreglos de tipo **caracter**, aplican todos los conceptos que ya explicaron para esta estructura de datos. Por otra parte, por el hecho de que las cadenas son de uso muy frecuente y generalizado en programación, en muchos lenguajes se dispone de muchas operaciones (funciones) sobre estas. La siguiente es una lista corta de estas operaciones para el pseudolenguaje, en la cual debe suponerse que **cad**, **cad1** y **cad2** son nombres de arreglos de tipo **caracter**.

OPERACIÓN	DESCRIPCIÓN
leerCadena(cad)	Guarda la cadena digitada por el usuario en el arreglo cad
escribirCadena(cad)	Escribe en la pantalla la cadena cad
longitudCadena(cad)	Retorna la longitud de la cadena cad
copiarCadena(cad1, cad2)	Copia la cadena cad2 en la cadena cad1
concatenarCadena(cad1, cad2)	Retorna la concatenación de cad1 con cad2 , en la cadena cad1
compararCadena(cad1, cad2)	Retorna menos uno (-1) si cad1 es menor que cad2 , cero (0) si son iguales y uno (1) si cad2 es menor que cad1

Para determinar si una cadena es menor que otra se usa el orden lexicográfico, es decir, el mismo que usan los diccionarios. Así por ejemplo, “casa” es menor “casita”, y esta a su vez es menor que “caza”. De otra parte, la operación **leerCadena** pone automáticamente el terminador de cadena, lo mismo que las operaciones **copiarCadena** y **concatenarCadena**.

2.1 Ejemplo completo.

Un **palíndromo** es una palabra o frase, si se quiere del idioma español, que se puede leer igual de izquierda a derecha y de derecha a izquierda, obviando signos de puntuación y espacios. Para aclarar, son palíndromos las siguientes frases y palabras:

- Anilina
- Amor a Roma
- Dábale arroz a la zorra el abad
- Reconocer
- Anita lava la tina
- ala

Considere el problema de construir un algoritmo que lea una PALABRA, de longitud máxima 30, y determine si es palíndromo o no.

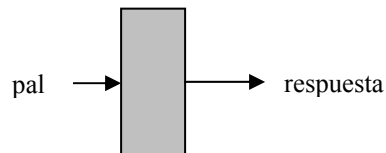
Las entradas (datos conocidos) para el algoritmo son:

- La palabra a considerar

La salida esperada (dato desconocido) es:

- Un mensaje que indica si es palíndromo o no

La siguiente gráfica resume las entradas y las salidas del algoritmo que se pretende diseñar. Además bautiza las variables principales:



Las condiciones iniciales y finales se pueden expresar mediante dos cláusulas: REQUIERE y GARANTIZA, de la siguiente manera:

REQUIERE:

Una palabra, llamada **pal**, que puede tener hasta 30 caracteres

GARANTIZA:

Escribe en pantalla un mensaje, **respuesta**, que indica si **pal** es palíndromo o no

Una primera versión del algoritmo solución puede ser simplemente la siguiente:

Inicio

Paso 1. Leer la palabra

Paso 2. Determinar si es palíndromo

Paso 3. Escribir en pantalla el mensaje apropiado

Fin

Los pasos 1 y 3 son interacciones con el usuario que se implementan usando las operaciones sobre cadenas disponibles. La versión inicial se puede refinar detallando estos pasos y además, definiendo las variables necesarias para hacerlo:

Procedimiento principal

Variables

pal: arreglo [31] de caracter

respuesta: arreglo [20] de caracter

Inicio

escribir("Por favor digite una frase de máximo 30 letras")

leerCadena(pal)

Paso 2. Determinar si pal es palíndromo

Paso 3. Escribir en pantalla el mensaje apropiado

Fin

Note que el arreglo **pal** se define de dimensión 31. Puede explicar cuál es el propósito de esto?.

La parte nuclear de la solución es el Paso 2 (determinar si el arreglo **pal** contiene un palíndromo). Para esto se deben comparar el primer carácter con el último, el segundo con el penúltimo, y así sucesivamente. En pseudolenguaje, este proceso puede ser:

```
i:= 0                                // i señala el primer caracter de la cadena
j:= longitudCadena(pal)-1// j señala el último caracter de la cadena
siga:= verdadero                     //variable que indica cuándo parar el proceso
mientras (i<j y siga) hacer
    si (pal[i]=pal[j]) entonces
        i:= i+1
        j:= j-1
    sino
        siga:= falso
    fin-si
fin-mientras
si i<j entonces
    copiarCadena(respuesta,"no es palindromo")
sino
    copiarCadena(respuesta,"es palindromo")
fin-si
```

El arreglo **respuesta** queda con la respuesta apropiada para el usuario.

El algoritmo completo se presenta enseguida. Se han definido algunas constantes para permitir que el programa sea más fácilmente modificable.

Procedimiento principal

Constantes

N 30 / N es la máxima longitud de la palabra dada por el usuario

Variables

pal: arreglo [31] de caracter

respuesta: arreglo [20] de caracter

i,j: entero

siga: booleano

Inicio

escribir("Por favor digite una palabra de máximo 30 letras")

leerCadena(pal)

i:= 0 // i señala el primer caracter de la cadena

j:= longitudCadena(pal)-1 // j señala el último caracter de la cadena

siga:= verdadero //variable "bandera" que indica cuándo parar el proceso

mientras (i<j y siga) hacer

si (pal[i]=pal[j]) entonces

i:= i+1

j:= j-1

sino

siga:= falso

fin-si

fin-mientras

```

si i<j entonces
    copiarCadena(respuesta,"no es palindromo")
sino
    copiarCadena(respuesta,"es palindromo")
fin-si
escribirCadena(respuesta)
Fin

```

2.2 Problemas

1. Construya un algoritmo que lea una FRASE, de longitud máxima 30, y determine si es palíndromo o no. El algoritmo debe procesar correctamente frases de más de una palabra.
2. Construya un algoritmo que lea una frase del español de máximo 100 caracteres y determine cuántas palabras, vocales y consonantes tiene.
3. Construya un algoritmo que lea dos palabras del español y determine si la primera es sufijo de la segunda. Por ejemplo, *lote* es prefijo de *casalote*.
4. Construya un algoritmo que lea dos palabras del español y determine si la primera es parte de la segunda. Por ejemplo, *alo* es parte de *casalote*.

2.3 Ejercicios

1. Suponga que la operación **longitudCadena** no está disponible. Escriba un algoritmo que calcule la longitud de una cadena, bajo el supuesto de que el terminador de cadena aparece en alguna casilla del arreglo.
2. Suponga que la operación **compararCadena** no está disponible. Escriba un algoritmo que compare dos cadenas, bajo el supuesto de que ambas tienen el terminador de cadena.
3. Escriba un algoritmo que invierta una cadena. Por ejemplo, si la cadena es *épica*, su inversa es *acipé*.

3. Matrices

Una matriz es una estructura de datos, o más técnicamente, un espacio de memoria que permite almacenar una colección de elementos, todos del mismo tipo. La diferencia con los arreglos está en que, en las matrices, los elementos no están organizados linealmente sino que su organización es bidimensional, es decir, en filas y columnas. Conviene imaginar una matriz como una organización de celdas de memoria, o **casillas**, en cada una de las cuales se puede guardar un elemento de la colección. Además, es usual dibujarla como lo ilustra la figura siguiente:

	0	1	2	3	4	5
0						
1						
2						
3						

Esta figura representa un matriz de cuatro filas (numeradas verticalmente de 0 a 3) y seis columnas (numeradas horizontalmente de 0 a 5). En cada una de las 24 celdas o casillas se puede guardar un dato. La **dimensión** o tamaño de una matriz es el número filas por el número de columnas. Debe ser claro entonces que la figura anterior es la gráfica de una matriz de dimensión 4x6.

La numeración de las filas y las columnas determina que cada una de las casillas de una matriz tiene asociados dos números que la identifican de manera única. A estos números se les llama **índice de fila** e **índice de columna**, respectivamente. En el pseudolenguaje, y también en C y C++, las filas y las columnas se numeran desde 0.

Los lenguajes como C y C++, permiten que el programador declare matrices de cualquier tipo y prácticamente de cualquier tamaño. En el pseudolenguaje, una matriz se declara usando el siguiente formato:

<NOMBRE> : matriz [<N>][<M>] de <TIPO>

En este formato aparecen en mayúsculas y entre los caracteres < y > los componentes que el programador puede determinar. Así por ejemplo, si se quiere declarar una matriz con nombre **mat**, de dimensión **15x4** y que pueda almacenar datos de tipo **caracter**, se debe escribir la siguiente línea.

mat : matriz [15][4] de caracter

Según el formato anterior, el programador debe bautizar la matriz (ponerle un nombre significativo), debe decir cuál es su dimensión, y también debe decir de qué tipo son los elementos que almacenará.

Enseguida se dan algunos ejemplos de declaraciones de matrices.

- Si se necesita guardar la información relacionada con el tablero de un juego de *tic tac toe* (el tradicional *triqui*), se puede declarar la siguiente matriz:

tablero : matriz [3][3] de caracter

- Si se requiere guardar las notas que han sacado 35 estudiantes en los 5 talleres y en los 5 laboratorios del curso de Programación de Computadores se pueden declarar las siguientes matrices.

talleres : matriz [35][5] de real
laboratorios : matriz [35][5] de real

Note que, en ambas matrices, cada fila guarda las notas de un estudiante del curso.

- Si se quiere guardar las letras que conforman una sopa de letras, como aquellas que vienen en los pasatiempos, se puede declarar la siguiente matriz.

sopa : matriz [10][15] de caracter

Note que la sopa de letras más grande que se puede guardar es de 10 filas por 15 columnas.

Los índices se crearon para permitir que el programador se pueda referir, de forma específica y directa, a una cualquiera de las casillas de la matriz, tanto para guardar un dato en esa casilla, como para obtener el dato almacenado en ella. En el pseudolenguaje, para referirse a una casilla particular de una matriz se debe seguir el siguiente formato:

<NOMBRE>[<INDICE-DE-FILA>][<INDICE-DE-COLUMNA>]

es decir, se debe escribir el nombre de la matriz seguido por el índice de fila y por el índice de columna, ambos entre paréntesis cuadrados, de la casilla que se quiere consultar.

Para los siguientes ejemplos, suponga que se declara la matriz **montos**, de la siguiente manera:

montos : matriz [6][10] de real

- La siguiente instrucción asigna o guarda el número 10,4 en la casilla de la esquina superior izquierda de esta matriz:

montos[0][0]:= 10,4

- La siguiente instrucción iterativa guarda 5,5 en cada una de las casillas de la última fila de esta matriz:

k:=0

MIENTRAS (k<10) HACER

montos[5][k]:= 5,5

k:=k+1

FIN-MIENTRAS

La siguiente figura muestra la matriz **montos** después de ejecutadas las instrucciones de los dos ejemplos anteriores. Las casillas vacías no tienen valores definidos.

	0	1	2	3	4	5	6	7	8	9
0	10,4									
1										
2										
3										
4										
5	5,5	5,5	5,5	5,5	5,5	5,5	5,5	5,5	5,5	5,5

3.1 Ejemplo completo

Una **matriz mágica** es una matriz cuadrada (tiene igual número de filas que de columnas) que tiene como propiedad especial que la suma de las filas, las columnas y las diagonales es igual. Por ejemplo:

2	7	6
9	5	1
4	3	8

En esta matriz las sumas son 15.

Considere el problema de construir un algoritmo que compruebe si una matriz de datos enteros es mágica o no, y en caso de que sea mágica escribir la suma. El usuario ingresa el tamaño de la matriz máximo hasta 10. Además debe guardar la suma de las filas, las columnas y las diagonales en un arreglo en el orden siguiente:

0	1	2		3	4	5		6	7
Fila 0	Fila 1	Fila 2	...	Columna 0	Columna 1	Columna 2	...	Diagonal 1	Diagonal 2

Las entradas (datos conocidos) para el algoritmo son:

- La dimensión de la matriz
- Los números que contiene la matriz

La salida esperada (datos desconocidos) es:

- La matriz es mágica o no, y si es mágica cuál es el valor de la suma.

En este problema, los arreglos son útiles para guardar los datos que conforman la matriz. Los números que contiene la matriz se pueden guardar en una variable entera.

La siguiente gráfica resume las entradas y salidas del algoritmo que se pretende diseñar. Además bautizan todas las variables mencionadas.

Se puede observar que el primer **ciclo para** tiene como contador la variable *i*, esto indica que se llenará la matriz por filas, el segundo **ciclo para** que tiene como contador la variable *j*, recorrerá la fila columna a columna para ubicar allí el dato correspondiente.

La parte nuclear de la solución es el paso 3. En este problema en particular se sabe que el número de filas y de columnas es igual y que hay dos diagonales. Para el ejemplo mostrado al inicio sería 3 filas, 3 columnas y dos diagonales. Para almacenar las sumas en un arreglo este tendrá una dimensión de $2 \cdot \text{tam} + 2$. La declaración del arreglo sumas es:

sumas: arreglo [22] de enteros

Ahora para calcular las sumas se puede hacer lo siguiente:

Paso 3.1: Inicializar el arreglo de sumas en cero

Paso 3.2: Sumar fila por fila, columna por columna y las diagonales y guardar su valor en el arreglo.

```
    para(i=0 hasta 2*tam+2) hacer
        sumas[i]:=0
    fin_para

//Sumas correspondientes a las filas
    para(i=0 hasta tam-1) hacer
        para(j=0 hasta tam-1) hacer
            sumas[i]=magica[i][j]+sumas[i]
        fin_para
    fin_para

//Sumas correspondientes a las columnas
    para(j=0 hasta tam-1) hacer
        para(i=0 hasta tam-1) hacer
            sumas[j+tam]=magica[i][j]+sumas[j+tam]
        fin_para
    fin_para

//Sumas correspondientes a las diagonales
    para(i=0 hasta tam-1) hacer
        sumas[2*tam]=magica[i][i]+sumas[2*tam]
    fin_para

    para(i=0 hasta tam-1) hacer
        sumas[2*tam+1]=magica[i][(tam-1)-i]+sumas[2*tam+1];
    fin_para
```

Paso 4: Para determinar si la matriz es mágica se va a recorrer y comparar el vector sumas, si en algún momento se encuentra un valor diferente se muestra en pantalla que la matriz no es mágica y se lleva el contador *i* más allá del final del arreglo, si por el contrario se llega al final del arreglo, es decir que todo este contiene el mismo valor y la matriz si cumple con las características evaluadas, se muestra en pantalla que la matriz es mágica.

//Comparar el vector suma y muestra el resultado

```

int con=0;
con=sumas[0];
para(i=1 hasta 2*tam+1) hacer
    si(con<>sumas[i])
        escribir("la matriz no es mágica")
        i=2*tam+3;
    fin_si
fin-para
si(i=2*tam+2)
    escribir("la matriz es mágica y la suma es:")
    escribir(con);
fin_si

```

El algoritmo completo se presenta enseguida.

Procedimiento principal

variables

```

i, j, aux, tam, suma: entero    //i señala las filas
                                //j señala las columnas

```

con=0: entero

magica: matriz [10][10] de enteros

sumas: arreglo [22] de enteros

Inicio

```

escribir("Por favor digite el número de filas de la matriz (entre 2 y 10): ")
leer(tam)

```

```

para (i=0 hasta tam-1) hacer
    para(j=0 hasta tam-1) hacer
        escribir("Por favor digite el dato en la posición")
        escribir(i,j)
        leer(magica[i][j])
    fin_para
fin_para

```

fin_para

fin_para

```

para(i=0 hasta 2*tam+2) hacer
    sumas[i]:=0

```

fin_para

//Sumas correspondientes a las filas

```

para(i=0 hasta tam-1) hacer
    para(j=0 hasta tam-1) hacer
        sumas[i]=magica[i][j]+sumas[i]
    fin_para
fin_para

```

fin_para

//Sumas correspondientes a las columnas

```

para(j=0 hasta tam-1) hacer
    para(i=0 hasta tam-1) hacer

```

```

        sumas[j+tam]=magica[i][j]+sumas[j+tam]
    fin_para
fin_para

//Sumas correspondientes a las diagonales
para(i=0 hasta tam-1) hacer
    sumas[2*tam]=magica[i][i]+sumas[2*tam]
fin_para

para(i=0 hasta tam-1) hacer
    sumas[2*tam+1]=magica[i][(tam-1)-i]+sumas[2*tam+1];
fin_para
con=sumas[0];
para(i=1 hasta 2*tam+1) hacer
    si(con<>sumas[i])
        escribir("la matriz no es mágica)
        i=2*tam+3;
    fin_si
fin-para
si(i=2*tam+2)
    escribir("la matriz es mágica y la suma es:")
    escribir(con);
fin_si
fin-procedimiento

```

3.1 Problemas

1. El dueño de un restaurante entrevista a cinco clientes de su negocio y les pide que califiquen de 1 a 10 los siguientes aspectos: (1 es pésimo y 10 es excelente o inmejorable)
 - Atención de parte de los empleados
 - Calidad de la comida
 - Justicia del precio (el precio que pagó le parece justo?)
 - Ambiente (muebles cómodos?, música adecuada?, iluminación suficiente?, decoración, etc.)

Escriba un algoritmo que pida las calificaciones de los cinco clientes a cada uno de estos aspectos, y luego escriba el promedio obtenido en cada uno de ellos. La lista debe aparecer ordenada del aspecto mejor calificado al peor calificado.

2. En una hacienda hay un hato que se compone de **N** vacas. Diseñe un algoritmo que guarde en una matriz de dimensión **7xN** la producción de leche diaria (en litros) de cada una de las vacas, durante una semana. Además, el algoritmo debe calcular la producción total del hato en cada uno de los siete días, y el número de la vaca que dio más leche en cada día.

3.2 Ejercicios

- Los siguientes ejercicios tienen como propósito que usted escriba ciclos que recorran la matriz completa o partes de ella. Suponga que se ha definido una constante positiva entera **N** y una matriz **mat**, de dimensión **NxN**.
 - Escriba un algoritmo que ponga cero en ambas diagonales de la matriz.
 - Escriba un algoritmo que ponga cero en la primera y la última fila, y en la primera y la última columna de la matriz.
 - Escriba un algoritmo que llene de números la matriz de tal forma que **mat[i][j]** sea igual a **i+j**.
 - Escriba un algoritmo que llene la diagonal principal de la matriz con los números **1,2,3,...N**. La diagonal principal de una matriz está formada por las casillas en las cuales el índice de fila y de columna son iguales.
 - Escriba un algoritmo que llene todas las filas pares con los números **1,2,3,...N**, y las filas impares con los números **N,N-1,N-2,...1**.
- Diseñe un algoritmo que permita guardar en un arreglo las sumas de las filas de una matriz. Esto es, la suma de los elementos de la primera fila deberá quedar guardada en la primera posición del arreglo, la suma de los elementos de la segunda fila en la segunda posición, y así sucesivamente para todas las filas de la matriz. **La máxima dimensión** de la matriz es 100x50 (100 filas y 50 columnas) y la del vector es 100.

Por ejemplo, si el usuario ingresa la siguiente matriz de 3x5 (3 filas, 5 columnas)

3,5	6,5	30	8,2	0
4	0	-1	3,6	1,4
10	-1,5	3,4	6,6	2

El resultado sería un arreglo siguiente:

48,2	8	20,5
------	---	------

porque

$$\begin{aligned} 3,5 + 6,5 + 30 + 8,2 + 0 &= 48,2 \\ 4 + 0 + (-1) + 3,6 + 1,4 &= 8 \quad \text{y} \\ 10 + (-1,5) + 3,4 + 6,6 + 2 &= 20,5 \end{aligned}$$

- En álgebra lineal las matrices son tema central. Sobre ellas se definen varias operaciones, como por ejemplo:
 - La suma de dos matrices. Si **A** y **B** son matrices de igual dimensión, la matriz **C=A+B** se calcula haciendo que **C[i][j] = A[i][j]+B[i][j]**, para todo **i** y **j** válidos.
 - La traspuesta de una matriz. Si **A** es una matriz de dimensión **NxM**, la matriz **B=A^t** se calcula haciendo que **B[i][j] = A[j][i]**, para todo **i** y **j** válidos. Note que esto quiere decir que las filas se convierten en columnas y que la dimensión de **B** es **MxN**.

- La traza de una matriz cuadrada. Si **A** es una matriz de dimensión **NxN**, la matriz traza es la suma de todos los elementos de la diagonal principal.
- La multiplicación de dos matrices. Si **A** y **B** son matrices de dimensiones **nxm** y **mxk**, respectivamente, la matriz **C=A*B**, de dimensión **nxk**, se calcula haciendo que:

$$C[i][j] = \sum_{p=0}^{m-1} A[i][p] * B[p][j]$$

Especifique y escriba un algoritmo para cada una de estas operaciones.

CODIFICACIÓN EN C++ DE ARREGLOS Y MATRICES

	Seudocódigo	C++
Arreglo	<NOMBRE> : arreglo [<N>] de <TIPO>	<TIPO> <NOMBRE>[<N>];
Cadena	<NOMBRE> : arreglo [<N>] de caracter	char <NOMBRE>[<N>];
Matriz	<NOMBRE> : matriz [<N>][<M>] de <TIPO>	<TIPO> <NOMBRE>[<N>][<M>];

EJEMPLO EN C++

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
```

/*Este programa lee los datos de una matriz 3x4 y muestra en pantalla la suma de los datos de cada fila */

```
main(){
    int matriz[3][4];
    int arreglo[3];
    int i,j;

    //Ingreso de los datos
    for (i=0;i<3;i++){
        for (j=0;j<4;j++){
            cout << "Ingrese el numero entero correspondiente a la posicion ["<i><i>"] [<j><j>]: ";
            cin >> matriz[i][j];
        }
    }
}
```

```

//Muestra en pantalla la matriz ingresada
cout << "\nLa matriz que usted ingreso es: \n\n";
for (i=0;i<3;i++){
    for (j=0;j<4;j++){
        cout << matriz[i][j]<<" ";
    }
    cout << "\n";
}

//Suma los datos de cada fila
for (i=0;i<3;i++){
    arreglo[i]=0;
    for (j=0;j<4;j++){
        arreglo[i]=arreglo[i]+matriz[i][j];
    }
}

//Muestra en pantalla los resultados
for (i=0;i<3;i++){
    cout << "\nLa suma de los datos de la fila "<<i<<" es: " << arreglo[i];
}
getch();
}

```